CRII: SHF: A New Foundation for Attack Trees Based on Monoidal Categories

Harley Eades III, Computer and Information Sciences,
Augusta University

1 Overview

Attack trees are a modeling tool, originally proposed by Bruce Schneier [31], which are used to assess the threat potential of a security critical system. Attack trees have since been used to analyze the threat potential of many types of security critical systems, for example, cybersecurity of power grids [36], wireless networks [30], and many others. Attack trees consists of several goals, usually specified in English prose, for example, "compromise safe" or "obtain administrative privileges", where the root is the ultimate goal of the attack and each node coming off of the root is a refinement of the main goal into a subgoal. Then each subgoal can be further refined. The leaves of an attack tree make up the set of base attacks. Subgoals can be either disjunctively or conjunctively combined.

Extensions of attack trees. There have been a number of extensions of attack trees to include new operators on goals. One such extension recasts attack trees into attack nets which have all of the benefits of attack trees with the additional benefit of being able to include the flaw hypothesis model for penetration testing [25]. A second extension adds sequential conjunction of attacks, that is, suppose A_1 and A_2 are attacks, then $A_1; A_2$ is the attack obtained by performing A_1, and then executing attack A_2 directly after A_1 completes [17].

The need for a foundation. Attack trees for real-world security scenarios can grow to be quite complex. The attack tree presented in [36] to access the security of power grids has twenty-nine nodes with sixty counter measures attached to the nodes throughout the tree. The details of the tree spans several pages of appendix. The attack tree developed for the border gateway protocol has over a hundred nodes [7], and the details of the tree spans ten pages. Manipulating such large trees without a formal semantics can be dangerous.

The formal semantics of attack trees. The leading question the field is seeking to answer by giving a mathematical foundation to attack trees is "what is an attack tree?" There have been numerous attempts at answering this question. For example, attack trees have been based on propositional logic and De Morgan Algebras [21, 20, 28], multisets [24], Petri nets [25], tree automata [5], and series parallel graphs [17]. **There is currently no known semantics of attack trees based in category theory**.

By far the most intuitive foundation of attack trees is propositional logic or De Morgan algebras, however, neither of these properly distinguish between attack trees with repeated subgoals. If we consider each subgoal as a **resource** then the attack tree using a particular resource twice is different than an attack tree where it is used only once. The multiset semantics of attack trees was developed precisely to provide a resource conscious foundation [24]. The same can be said for the Petri nets semantics [25]. A second benefit of a semantics based in multisets, Petri nets, and even tree automata is that operators on goals in attack trees are associated with concurrency operators from process algebra. That is, the goals of an attack tree should be thought of as being run concurrently – it seems this connection to process algebra has been overlooked. Furthermore, when moving to these alternate foundations **the intuitiveness and elegance of the propositional logic semantics is lost**. Lastly, existing work has focused on specifically what an attack tree is, and has not sought to understand what the theory of attack trees is.

Category Theory. Each of the various existing mathematical foundations attempt to answer the question "what is an attack tree?", but they are all seemingly very different mathematical structures. Is there a unifying core foundation common to all of these existing foundations? A categorical foundation will answer this question in the positive. The powers of category theory – an abstract branch of mathematics first proposed by Samuel Eilenberg and Saunders Mac Lane [23] – are its ability to abstract away unneeded details from a mathematical structure, and its ability to form relationships between seemingly unrelated mathematical structures. For example, intuitionistic logic and the λ-calculus both correspond to cartesian closed categories, and hence, are different perspectives of the same theory [22]. A categorical foundation of attack trees will result in the most basic mathematical foundation of attack trees, furthermore, it will reveal a relationship between all of the existing mathematical foundations, and lastly, it will reconnect attack trees with logic, but also forge new connections with functional programming languages and formal verification through the Curry-Howard-Lambek correspondence. This implies that by giving attack trees a semantics in category theory one obtains a more powerful semantic analysis and the ability to **derive a programming language that can not only define attack trees, but also reason about them using formal verification** (Section 3). In addition, the various graphical languages used in category theory, [33], may lead to new graphical tools for threat analysis.

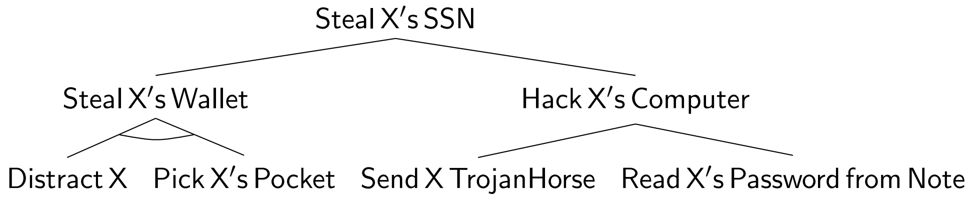
This Proposal. I propose to found attack trees in linear logic rather than propositional logic by giving attack trees a semantics in symmetric monoidal categories. A semantics in symmetric monoidal categories is a generalization over the previous forms of models of attack trees. Multisets and Petri nets are examples of symmetric monoidal categories [3, 37]. Furthermore, symmetric monoidal categories are a categorical model of linear logic [9], thus regaining the elegant connection between attack trees and logical formulas. The connection to linear logic also opens the door to a connection between threat analysis using attack trees and process algebra such as Petri nets, but also Chu spaces [29], dialectica spaces [8], and session types [38, 4]. I and my trainees will fully develop the semantics of attack trees in symmetric monoidal categories and show that not only can attack trees be modeled by these types of categories, but so can a range of their extensions like attack trees with sequential conjunction. We will also investigate new attack tree operators based on the connection to process algebra. Then we will develop a new formal system called Lina for Linear Threat Analysis (Section 3). Lina will be a statically typed linear polymorphic functional domain-specific programming language designed to construct, manipulate, and prove properties of attack trees. Lina will represent attack trees as linear types, and thus, programs between these types will be considered transformations of attack trees. The semantics of Lina will also be in symmetric monoidal categories forming a tight correspondence with the semantics of attack trees. This system will be the first threat analysis tool to support proving properties of attack trees, thus **connecting software verification to threat analysis.** Finally, new threat analysis tools can be built on top of Lina.

Outcomes and impact. By basing attack trees in linear logic this work will connect threat analysis to resource conscious logics, software verification and process algebra. This will open the door for many interesting future projects in all of these areas of study. This work will also provide new extensions of attack trees and new extensions of linear logic, both of which will have a significant impact on the field. Lina's representation of attack trees as types and transformations as programs is a completely novel representation of attack trees. Furthermore, Lina has practical significance due to its ability to not only define and manipulate attack trees, but actually prove properties about attack trees.

PI qualifications. I have extensive experience designing and analyzing both simply typed and dependently typed programming languages and logics. As part of my dissertation I was a member of the Trellys project [6, 19, 35, 34] in which I contributed to the design of a dependently typed functional programming language called Separation of Proof from Program (Sep^3) whose design is now detailed in my dissertation [14, 19]. The second major part of my dissertation was dedicated to the meta-theoretic analysis of various type theories. First, I studied a new proof technique for proving normalization of simply typed and predicative polymorphic type theories called hereditary substitution, and then proved completeness of a new logic and simple type theory called dualized simple type theory [14].

2 Linear Attack Trees in Symmetric Monoidal Categories

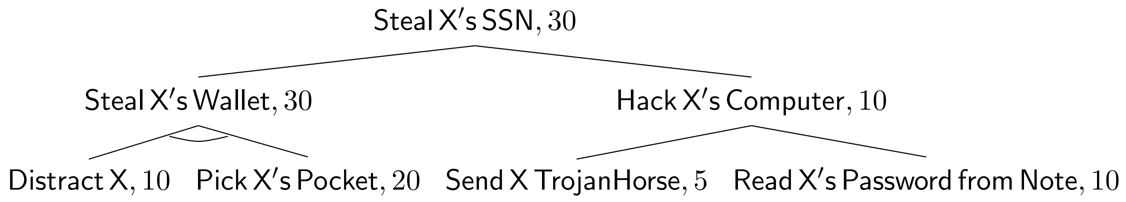
In this section we give some preliminary work on giving attack trees a semantics in symmetric monoidal categories. This semantics is the starting point for the Lina programming language implementation. Suppose we wanted to assess the threat of whether an attacker could steal the social security number of person X. An attack tree could be used and might look something like the following:



The root of the attack tree is the goal of the attack, and then this goal is refined into two new subgoals which are to either steal X's wallet or to hack their computer. Then these two goals can be refined further. Note that there is an arc between the two edges connecting the subgoals of the goal to steal X's wallet. This arc signifies that the two subgoals must be done together. These are often called conjunctive subgoals while all of the other goals are disjunctive. The graphical presentation of attack trees is very appealing, but for large complex systems it becomes hard to utilize. So attack trees are often presented in the form of a script. A third representation of attack trees uses n-ary operators; for example, we can represent the tree from above by the following:

$$\begin{array}{rcl}
\mathsf{Steal\ X's\ SSN} & = & (\mathsf{Steal\ X's\ Wallet}) + (\mathsf{Hack\ X's\ Computer}) \\
\mathsf{Steal\ X's\ Wallet} & = & (\mathsf{Distract\ X}) \otimes (\mathsf{Pick\ X's\ Pocket}) \\
\mathsf{Hack\ X's\ Computer} & = & (\mathsf{Send\ X\ TrojanHorse}) + (\mathsf{Read\ X's\ Password\ from\ Note})
\end{array}$$

After developing an attack tree it is common to add a weight to each node of the tree. This weight can be thought of as the cost of conducting that attack. Then attacks with high cost can be considered as unlikely and safely ignored. For example, we can give weights to the attack tree given above:



After giving attacks costs different projections can be conducted on the attack tree to project out attacks meeting some criteria. We concentrate on attack trees without weights or any additional structures on the nodes here.

At this point we give the formal definition of a new form of attack tree we call linear attack trees (or just attack trees).

Definition 1. ***Linear Attack trees*** *are defined by the following grammar:*

$$\begin{array}{lcl} B & ::= & b_1 \mid \cdots \mid b_n \\ A & ::= & B \mid \otimes(A,\ldots,A) \mid \times(A,\ldots,A) \mid +(A,\ldots,A) \mid ©(A) \end{array}$$

We call B the set of base attacks, and we will often treat the attack tree operators as infix operators. The following rules define the attack tree reduction relation:

$$\frac{}{(A \otimes B) \otimes C \rightsquigarrow A \otimes (B \otimes C)}\ \text{ASSOC} \qquad \frac{}{A \otimes (B + C) \rightsquigarrow (A \otimes B) + (A \otimes C)}\ \text{DIST}$$

The previous rules can be applied on any well-formed subattack tree. The equivalence relation, denoted $\equiv$, on attack trees is defined as the reflexive, symmetric, and transitive closure of the reduction relation $\rightsquigarrow$.

The two rules we consider here, association and distribution, are similar rules considered by Mauw and Oostdijk [24]. One of the novel aspects of this work is we treat each attack as a process and the attack tree operators as concurrency operators on processes. Now suppose A_1 and A_2 are two attack trees. Then the attack tree defined by $A_1 \otimes A_2$ is the process where attack A_1 is run parallel to attack A_2, this is called parallel composition of attack trees. The attack tree $A_1 \times A_2$ corresponds to the process that executes attacks A_1 and A_2 in any order. The attack tree that executes attacks A_1 or A_2 in any order is defined by $A_1 + A_2$. This interpretation suggests that attacks are resources and as such cannot be freely duplicated or deleted, and thus, we add an explicit operator, $©(A)$, which should be read as "A can be copied." Our semantics will enforce that the only attack trees that can be copied must be under the copy operator.

Our goal of this section is to give a semantics of attack trees, as defined above, in symmetric monoidal categories[1] with some additional structure.

Definition 2. *A* ***symmetric monoidal category****, $(\mathbb{C}, I, \otimes)$, is a category with a binary endofunctor $\otimes : \mathbb{C} \times \mathbb{C} \longrightarrow \mathbb{C}$, an object $I \in \mathsf{Obj}(\mathbb{C})$, natural isomorphisms $\lambda_A : I \otimes A \longrightarrow A$, $\rho_A : A \otimes I \longrightarrow A$, and $\alpha_{A,B,C} : A \otimes (B \otimes C) \longrightarrow (A \otimes B) \otimes C$, and a natural transformation $\beta_{A,B} : A \otimes B \longrightarrow B \otimes A$. The natural isomorphisms and transformations are subject to several coherence equations that we omit; see [23].*

[1]Suppose $f : A \longrightarrow B$ and $g : B \longrightarrow C$ are two arbitrary morphisms of some category $\mathbb{C}$, then we denote their composition by $f; g : A \longrightarrow C$.

Basic symmetric monoidal categories are enough to model attack trees which solely consist of the parallel composition operator $\otimes$, and thus, to model the other operators we must extend monoidal categories with some additional structure.

Definition 3. *An* ***additive linear category****,* $(\mathbb{C}, I, \otimes, 1, 0, \times, +)$ *is a symmetric distributive monoidal category,* $(\mathbb{C}, I, \otimes)$*, with products and coproducts. Being distributive means that for any objects* $A, B, C \in \mathsf{Obj}(\mathbb{C})$*, there is a natural isomorphism,* $\mathsf{d}_{A,B,C} : A \otimes (B + C) \longrightarrow (A \otimes B) + (A \otimes C)$*.*

At this point additive linear categories contain enough structure to model all of attack trees, but the copy operator.

Definition 4. *A* ***linear attack model*** *consists of the following data:*

- *An additive linear category,* $(\mathbb{L}, I, \otimes, 1, 0, \times, +)$*,*
- *a distributive category*[2]*,* $(\mathbb{C}, 1, 0, \times, +)$*,*
- *a pair of symmetric monoidal functors* $F : \mathbb{C} \longrightarrow \mathbb{L}$ *and* $G : \mathbb{L} \longrightarrow \mathbb{C}$*, such that,* F *and* G *form a symmetric monoidal adjunction*[3]*.*

The previous definition is based on the notion of a linear/non-linear model of intuitionistic linear logic [1]. Intuitively, think of the category $\mathbb{L}$ as the model of resource conscious attack trees, and the category $\mathbb{C}$ as the model of attack trees based on propositional logic [21, 20]. Then the adjoint functors F and G are translations between the two types of models. These do not create an isomorphism, but only a relationship.

An analogy might help with understanding how to view the adjunction. Suppose there is an Earth like planet called planet X. Now planet X is just like Earth, but it has a slightly less powerful force of gravity. This implies that all of Earth's physics can be translated to the physics of planet X by weakening gravity, and planet X's physics can be translated to Earth's physics by strengthening gravity. Thus, we have an adjunction between planet X's physics and Earth's physics.

We will see that the composition of F and G will model the copy operator. The beautiful part of this model over existing models is that it can handle attack trees based on resources, attack trees based propositional logic, and a hybrid of both. The copy operator gives us the best of both worlds.

Definition 5. *Suppose* B *is a set of basic attacks, and* $(\mathbb{L}, \mathbb{C}, G, F)$ *is a linear attack model,* $\mathsf{ob} : B \longrightarrow \mathsf{Obj}(\mathbb{L})$ *is an injective function, and* $\mathsf{op} \in \{\otimes, \times, +\}$*. Then we define the interpretation of attack trees as objects of* $\mathbb{L}$ *by induction as follows:*

$$\begin{array}{lcl} [\![b]\!] & = & \mathsf{ob}(b) \\ [\![A_1 \mathbin{\mathsf{op}} A_2]\!] & = & [\![A_1]\!] \mathbin{\mathsf{op}} [\![A_2]\!] \\ [\![\copyright A]\!] & = & F(G([\![A]\!])) \end{array}$$

We will now denote $G;F$ *by* $\copyright$*. The interpretation of the reduction relation for attack trees as morphisms of* $\mathbb{L}$ *is as follows:*

[2]A cartesian category with coproducts where products distribute over coproducts

[3]For the definitions of symmetric monoidal functor/adjunction see [1].

$$\begin{array}{lcl} [\![(A \otimes B) \otimes C \rightsquigarrow A \otimes (B \otimes C)]\!] & = & \alpha^{-1}_{[\![A]\!],[\![B]\!],[\![C]\!]} \\ [\![A \otimes (B + C) \rightsquigarrow (A \otimes B) + (A \otimes C)]\!] & = & \mathsf{d}_{[\![A]\!],[\![B]\!],[\![C]\!]} \end{array}$$

There are two main points that one should take away from the previous definition. The first is that the attack trees are simply objects of the category, but even more so, the morphisms are semantically valid transformations on the attack trees. Finally, the second point is that the copy operator can be seen as a gateway traveling from linear logic (monoidal category) to propositional logic (cartesian category). This then implies that proofs of equivalences of two attack trees corresponds to proofs of isomorphisms between morphisms of the model. In fact, it is straightforward to prove the following.

Lemma 6 (Soundness)**.** *For any attack trees A and B, if $A \equiv B$, then $[\![A]\!] = [\![B]\!]$.*

Linear attack models as defined above are left abstract, but our interpretation shows that any concrete mathematical structure that can be proven to be an attack model is sufficient to use to reason about attack trees. For example, Petri nets have been used to model attack trees [25], but Petri nets can be modeled by dialectica categories [3] which themselves have been shown to be an example of a linear/non-linear model of linear logic, and thus, essentially corresponds to a linear attack model [16].

3 Proposed: The Lina Implementation

My trainees and I will develop an implementation of a system for specifying and reasoning about attack trees called Lina for Linear Threat Analysis. Lina will consist of a core language and a surface language. The core language will include a decidable type checker using term annotations on types; note that type checking becomes undecidable in the presence of parametric polymorphism [40]. Programming with annotations can be very cumbersome, and so the surface language will use local type inference [27] to alleviate some of the burden from annotations. However, the surface language will provide further conveniences in the form of automation, to be used with labeled attack trees, and graphical representations of attack trees based on the various graphical languages used in category theory [33].

3.1 The Lina Core Language

The core language of Lina will have the tightest correspondence with the attack model semantics of attack trees. It will be an extension of Benton's linear/non-linear term assignment [1]. The extensions will include second-order polymorphism, linear additive products and linear additive coproducts. The addition of second-order polymorphism raises the issue of extending our attack model to ensure that the correspondence with Lina is upheld. We conjecture that this will be possible by extending our attack model to a hyperdoctrine model in a similar way to Seely's hyperdoctrine model for classical linear logic [32]. A last extension from Benton's term assignment is that we will add type annotations to terms to make type checking decidable.

Labeled attack trees. So far the core language has mainly been concerned with the shape of attack trees, and whether or not one attack tree is the same as another attack tree with a different shape. We will add the ability to label attack trees with weights and do projections on attack trees. There is one hurdle that needs to be addressed and that is whether or not Lina should allow reasoning between attack trees with labels. This extension might result in having to extend the equivalence relation on attack trees to account for labels. I conjecture that by layering a semi-ring over the attack model we could extend the core language with abstract labels to facilitate this addition. This would be similar to how Maw and Oostdijk [24] handle labels in their model.

Verification. The Lina core language can be seen as essentially being derived from the attack model using the Curry-Howard-Lambek correspondence. Objects will correspond to types, and morphisms will correspond to well-typed programs. Thus, defining an attack tree will correspond to defining a type. Then performing semantically valid transformations on attack trees, say from attack tree A_1 to attack tree A_2, corresponds to defining a program whose input type is A_1 and whose output type is A_2. This program can be seen as evidence that A_2 is semantically related to A_1, and thus, can be seen as a proof that A_1 implies A_2. In fact, we could prove that A_1 and A_2 are equivalent if we can write programs from A_1 to A_2 and vice versa. This type of verification is called interactive theorem proving and it is the locus of proof in interactive theorem provers such as Coq [10] and Agda [26].

Combing this with polymorphism allows for the construction of generic attack tree transformations. In this way we can view type variables as holes that can be filled with attack trees. Thus, providing a mechanism for defining new attack tree operators within Lina without necessarily having to extend the core language.

3.2 The Lina Surface Language

The Lina surface language will provide a more user-friendly programming environment by alleviating the burden of programming with type annotations using local type inference, and by providing a means of writing mutually recursive functions much like one would find in Haskell and Agda. Attack trees are types, and these types can become quite large, so to facilitate their construction the surface language will also provide type-level recursive global definitions.

A graphical language. We will also develop a web based graphical interface for proving equivalences of attack trees. For example, suppose we wanted to prove that the attack tree $A \otimes (B \otimes C)$ is equivalent to $(A \otimes B) \otimes C$. Then instead of providing the proof explicitly using the programming constructs one can provide a proof using an alternative graphical calculus that then translates to the traditional surface language. The graphical language will support the definition of an attack tree by placing nodes and edges on a canvas, but in addition it will provide the user with several predefined graphical gadgets that describe equations between the shapes of two subtrees. In the core language these gadgets are represented by isomorphisms between types that represent the subtrees. The user can use these graphical equations to conduct semantically valid transformations on attack trees in a completely graphical fashion. I conjecture this to be possible by exploiting the work of Blute et al. [2] and/or the work on open graphs used by the Quantomatic application [12, 11].

4 Personnel, Equipment, and Work Plan

Personnel. I am requesting funds to support six undergraduates and two Ph.D. students. The Ph.D. students will be hired during the summers of the first and second years. During the first year, the Ph.D. student will aid me in extending the linear attack tree semantics to include more attack tree operators, and the student during the second year will aid me in conducting the meta-theoretic analysis of Lina.

The undergraduate employees during the first year will aid me in developing the Lina core language. Then in the second year the undergraduate students will aid me in developing the surface language. More precisely, the undergraduate students during the second year will develop the graphical interface to the surface language.

Augusta University does not at this time have a graduate program in computer science, and thus, the Ph.D. students recruited to work with me will come from another university. During the summer of 2015 Augusta University established the Cyber Institute. It is the goal of the institute as well as the computer and information sciences department to eventually add a graduate program in computer science. Visiting Ph.D. students will contribute to this eventual goal. This would also be a valuable experience for the Ph.D. students, because they would get the chance to work with a researcher outside of their university. The Ph.D. students will need a basic foundation in either category theory or the meta-theoretic analysis of functional programming languages. To recruit these students I plan to advertise the opportunity to US based universities.

I plan to recruit the undergraduates who will aid me in the implementation of the Lina core and surface languages from my upper-level programming languages course. In this course I teach the basics of the design and implementation of functional programming languages using Haskell. So these students will already have the necessary introduction to Haskell, the language we will be developing Lina in, and the implementation of programming languages needed for the project.

The undergraduates that will aid me in implementing the graphical interface to the Lina surface language will not need to know Haskell or how to implement functional programming languages, but will need a foundation in web development. I plan to recruit these students from the pool of students who have taken the web development course that is taught in my department.

Equipment. To develop and host the Lina graphical web application I am asking for the funds to buy and host a development server. It will be hosted and maintained by the GRU Cyber Institute at Augusta University.

5 Broader Impacts of the Proposed Work

The primary impact of this work is in the advancement of the state of the art in the mathematical foundations of threat analysis and the tool support for threat analysis. The additional impact is in mentoring and teaching.

5.1 Advances over Previous Work

The mathematical foundations of attack trees proposed here is the first foundation based in category theory and linear logic. Past mathematical foundations are either very complex, or do not treat goals as resources. In Section 2 we give the first mathematical foundation of attack trees that provides both a semantics for a resource conscience interpretation simultaneously with an interpretation in propositional logic. It is possible to have the best of both worlds! By founding attack trees in category theory we are able to derive a core programming language (Section 3.1) with the ability to define and reason about attack trees. Then by creating a surface language over the core we will have the first threat analysis tool that is semantically correct, and with the ability to not only define and manipulate attack trees, but verify properties about them formally. Lastly, this work is the first to apply formal verification to threat analysis.

5.2 Teaching and Mentoring

Teaching. I plan to create a section on domain specific programming languages in my upper-level undergraduate programming languages course where my students will explore the design of Lina and conduct some case studies using Lina.

Undergraduate Research. I am the first person in my family to go to college, and I was well within my twenties when I entered my bachelor's program making me a non-traditional student. This is not an uncommon story of the vast majority of the students at Augusta University. Most are U.S. native non-traditional or even post-baccalaureate students. Due to the close proximity to Fort Gordon a large percentage of our students are either active military or military veterans.

I am dedicated to working with non-traditional students – 30% of undergraduate students enrolled at Augusta University are non-traditional students (25 years of age or older) – military veterans – 9% of undergraduate students at Augusta University are either military verterns, active military, or are members of a military family – and people of minority groups – 32% of undergraduate students at Augusta University are people of minority groups – by providing them with rich research experiences. Undergraduate research opportunities at my university have been sparse, and so this proposal will contribute to creating more research possibilities for these types of students.

Undergraduate students working on a research project like the one being proposed here should get a chance to do more than the expected tasks, for example, just writing programs. I want to include the students in all aspects of the project including the design and theoretical aspects of the project. This will give them a chance to really experience research. Furthermore, I want them to get the experience of writing up the research, and so they will be included in the final stages of the project as well. This experience could help convince the types of students listed above to pursue a career in research and go to graduate school when they would otherwise never think that such a career path was possible for them.

Graduate Research. At this time Augusta University does not have a graduate program in computer science. However, the Computer and Information Sciences department does have plans of adding one in the future. The visiting Ph.D. students will aid in helping these plans move forward by sparking interest in our department at other schools, but by also giving the department an opportunity to work with the visiting graduate students.

In addition, the visiting graduate students will help spark undergraduate interest in attending graduate school. Both types of students will have a chance to work together, where otherwise the students at Augusta University would not have such a chance to gain a first hand perspective of graduate school and research. Lastly, the visiting graduate students will have a chance to experience helping mentoring undergraduate students in a research project.

6 Prior NSF Support

During graduate school I was a contributing member of the NSF funded Trellys project which was a collaboration between Aaron Stump and his students at the University of Iowa, Stephanie Weirich and her students at the University of Pennsylvania, and Tim Sheard and his students at Portland State University to design a new general purpose dependently-typed functional programming language that supports type-based verification. The primary aim of the project was understanding how to mix type-based verification with general recursion.

My primary contribution to this project was to the design and analysis of the core language. The design space for the core language is quite large and as a team we made the decision to investigate three different language designs, one by each school. At the University of Iowa we designed a language called Separation of Proof and Program (Sep3) [19, 18, 14]. It is well known that if one designs their richly typed functional programming language such that every program defined within it terminates, then one can consider the programs as proofs and their types predicates. Thus, verification of programs turns into writing other programs viewed as proofs. However, with respect to Trellys, this design no longer applies, because we want to have general recursion; hence, not every program terminates. Sep3 overcomes this by separating the two worlds. Then, we link the two fragments together, so that the logical fragment can verify properties of the programs written inside the programmatic fragment.

The second part of my contribution to the Trellys project was to the meta-theoretic analysis of the logical fragment of the Trellys core language. My advisor and I investigated the potential of using a technique known as hereditary substitution – due to Watkins et al. [39] – to prove weak normalization of the logical fragment. Hereditary substitution is a function that substitutes the arguments of functions for parameters, but if a new function call is created it hereditarily reduces it. Now it turns out that to show the logical fragment is terminating it suffices to show the hereditary substitution function is terminating. This technique is simpler than other known techniques, and is easier to formalize in a proof assistant. We were the first to apply hereditary substitution to show weak normalization of predicative-polymorphic programs, and we were the first to extend this technique to programming languages with control operators [15, 13, 14].

The project was very collaboration based and I got the chance to collaborate across schools. Annually for the lifetime of the project all of the contributors of the project would gather at one of the three universities. During these annual meetings we would present all of our recent results, and work closely on problems we have encountered and work together to get past those problems. I had the pleasure to visit and work with Tim Sheard and his students at Portland State University. Lastly, I spent the summer of 2013 working closely with Stephanie Weirich and her students in which we worked on extending dependent types to classical type theories and control operators. In addition, during my visit I presented my thesis work on hereditary substitution to the Penn PL club. These collaborations were hugely important for my education and research, and I want to give similar experiences to my current and future students. This proposal has the potential to begin offering such experiences.

References

[1] Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models (preliminary report). Technical Report UCAM-CL-TR-352, University of Cambridge Computer Laboratory, 1994.

[2] R. F. Blute, J.R.B. Cockett, R.A.G. Seely, and T. H. Trimble. Natural deduction and coherence for weakly distributive categories. *Journal of Pure and Applied Algebra*, 113(3):229–296, 1996.

[3] Carolyn Brown, Doug Gurr, and Valeria Paiva. A linear specification language for petri nets. *DAIMI Report Series*, 20(363), 1991.

[4] LuÃ-s Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In Paul Gastin and FranÃ§ois Laroussinie, editors, *CONCUR 2010 - Concurrency Theory*, volume

6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer Berlin Heidelberg, 2010.

[5] S.A. Camtepe and B. Yener. Modeling and detection of complex attacks. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 234–243, Sept 2007.

[6] Chris Casinghino, Vilhelm Sjöberg, and Stephanie Weirich. Step-indexed normalization for a language with general recursion. In *Proceedings Fourth Workshop on Mathematically Structured Functional Programming*, 2012.

[7] S. Convery, D. Cook, and M. Franz. An attack tree for the border gateway protocol. 2003. `https://tools.ietf.org/html/draft-ietf-rpsec-bgpattack-00`.

[8] Valeria de Paiva. Dialectica and chu constructions: Cousins? *Theory and Applications of Categories*, 17(7):127–152, 2006.

[9] Valeria de Paiva. Categorical semantics of linear logic for all. In Luiz Carlos Pereira, Edward Hermann Haeusler, and Valeria de Paiva, editors, *Advances in Natural Deduction*, volume 39 of *Trends in Logic*, pages 181–192. Springer Netherlands, 2014.

[10] The Coq development team. The coq proof assistant referenence manual. LogiCal Project, 2015. Version 8.4.

[11] Lucas Dixon, Ross Duncan, and A. Kissinger. Open graphs and computational reasoning. *Extended Abstract: Developments in Computational Models*, 2010.

[12] Lucas Dixon and Aleks Kissinger. Open-graphs and monodial theories. May 2011.

[13] Harley D. Eades III and Aaron Stump. Hereditary substitution for the $\lambda\Delta$-calculus. In Ugo de'Liguoro and Alexis Saurin, editors, Proceedings First Workshop on *Control Operators and their Semantics,* Eindhoven, The Netherlands, June 24-25, 2013 , volume 127 of *Electronic Proceedings in Theoretical Computer Science*, pages 45–65. Open Publishing Association, 2013.

[14] Harley D. Eades III. *The Semantic Analysis of Advanced Programming Languages*. PhD thesis, University of Iowa, 2014.

[15] Harley D. Eades III and Aaron Stump. Hereditary substitution for stratified system f. *Proof-Search in Type Theories (PSTT)*, 2010.

[16] Harley Eades III and Valeria de Paiva. Multiple conclusion intuitionistic linear logic and cut elimination. `http://metatheorem.org/papers/FILL-report.pdf`.

[17] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, SaÅ!'a RadomiroviÄ, and Rolando Trujillo-Rasua. Attack trees with sequential conjunction. In Hannes Federrath and Dieter Gollmann, editors, *ICT Systems Security and Privacy Protection*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 339–353. Springer International Publishing, 2015.

[18] Garrin Kimmell, Aaron Stump, Harley D. Eades III, Peng Fu, Tim Sheard, Stephanie Weirich, Chris Casinghino, Vilhelm Sjoberg, Nathan Collins, and Ki Yung Ahn. Equational reasoning about programs with general recursion and call-by-value semantics. *Programming Languages Meets Program Verifcation (PLPV)*, 2012.

[19] Garrin Kimmell, Aaron Stump, Harley D. Eades III, Peng Fu, Tim Sheard, Stephanie Weirich, Chris Casinghino, Vilhelm Sjoberg, Nathan Collins, and Ki Yung Ahn. Equational reasoning about programs with general recursion and call-by-value semantics. *Special issue of Progress in Informatics*, March 2013.

[20] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. Computational aspects of attack–defense trees. In Pascal Bouvry, MieczysławA. Kłopotek, Franck Leprévost, Małgorzata Marciniak, Agnieszka Mykowiecka, and Henryk Rybiński, editors, *Security and Intelligent Information Systems*, volume 7053 of *Lecture Notes in Computer Science*, pages 103–116. Springer Berlin Heidelberg, 2012.

[21] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. A probabilistic framework for security scenarios with dependent actions. In Elvira Albert and Emil Sekerinski, editors, *Integrated Formal Methods*, volume 8739 of *Lecture Notes in Computer Science*, pages 256–271. Springer International Publishing, 2014.

[22] J. Lambek. From lambda calculus to cartesian closed categories. *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 376–402, 1980.

[23] Saunders Mac Lane. *Categories for the Working Mathematician.* Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971.

[24] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In DongHo Won and Seungjoo Kim, editors, *Information Security and Cryptology - ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 186–198. Springer Berlin Heidelberg, 2006.

[25] J. P. McDermott. Attack net penetration testing. In *Proceedings of the 2000 Workshop on New Security Paradigms*, NSPW '00, pages 15–21, New York, NY, USA, 2000. ACM.

[26] Ulf Norell. Dependently typed programming in agda. In *Proceedings of the 4th international workshop on Types in language design and implementation*, TLDI '09, pages 1–2, New York, NY, USA, 2009. ACM.

[27] Benjamin C. Pierce and David N. Turner. Local type inference. *ACM Trans. Program. Lang. Syst.*, 22(1):1–44, January 2000.

[28] L. Piètre-Cambacédès and M. Bouissou. Beyond attack trees: Dynamic security modeling with boolean logic driven markov processes (bdmp). In *Dependable Computing Conference (EDCC), 2010 European*, pages 199–208, April 2010.

[29] Vaughan Pratt. Chu spaces. Notes for the School on Category Theory and Applications University of Cimbra, July 1999.

[30] A. Reinhardt, D. Seither, A. Konig, R. Steinmetz, and M. Hollick. Protecting ieee 802.11s wireless mesh networks against insider attacks. In *Local Computer Networks (LCN), 2012 IEEE 37th Conference on*, pages 224–227, Oct 2012.

[31] Bruce Schneier. Attack trees: Modeling security threats. *Dr. Dobb's journal*, December 1999.

[32] R.A.G. Seely. Polymorphic linear logic and topos models. *C.R. Math. Rep. Acad. Sci. Canada*, XII(1), February 1990.

[33] Peter Selinger. A survey of graphical languages for monoidal categories. *ArXiv e-prints*, August 2009.

[34] Vilhelm Sjoberg, Chris Casinghino, Ki Yung Ahn, Nathan Collins, Harley D. Eades III, Peng Fu, Garrin Kimmell, Tim Sheard, Aaron Stump, and Stephanie Weirich. Irrelevance, heterogeneous equality, and call-by-value dependent type systems. In James Chapman and Paul Blain Levy, editors, Proceedings Fourth Workshop on *Mathematically Structured Functional Programming,* Tallinn, Estonia, 25 March 2012, volume 76 of *Electronic Proceedings in Theoretical Computer Science*, pages 112–162. Open Publishing Association, 2012.

[35] Vilhelm Sjöberg and Stephanie Weirich. Programming up to congruence. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 369–382, New York, NY, USA, 2015. ACM.

[36] Chee-Wooi Ten, Chen-Ching Liu, and Manimaran Govindarasu. Vulnerability assessment of cybersecurity for scada systems using attack trees. In *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–8, June 2007.

[37] A Tzouvaras. The linear logic of multisets. *Logic Journal of IGPL*, 6(6):901–916, 1998.

[38] Philip Wadler. Propositions as sessions. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, pages 273–286, New York, NY, USA, 2012. ACM.

[39] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework: The propositional fragment. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs*, volume 3085 of *Lecture Notes in Computer Science*, pages 355–377. Springer Berlin / Heidelberg, 2004.

[40] J.B. Wells. Typability and type checking in system f are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1â3):111 – 156, 1999.